

XDevCamp

Magento Workshop 2

Performance & Development

Vinai Kopp

Setup and Configuration

The workshop is designed to be followed on a Magento CE 1.6 Installation with the custom example modules **Example_Brand** and **Example_MultiOptionCart**. For Innovate attendees a VirtualBox image is available that contains a preconfigured environment.

If you prefer to use your own development environment, you may install the tgz archive of the source and the DB dump.

The code for the hands-on labs is available in pre-baked snippets in a separate archive file for your convenience.

Workshop Resources:

- A. VirtualBox VM Image (includes all workshop assets)
Magento_Workshop.vdi
- B. Magento Workshop Files Archive (Magento, the pre-baked code and the DB-dump)
Magento_Workshop_Full.tgz
- C. Archive containing pre-baked code snippets
Magento_Workshop2_Code.tgz

Preparations with the Innovate VirtualBox Image:

- 1. Fire up the VirtualBox VM using the supplied disk image
- 2. Open the Firefox Browser (a shortcut button is on the desktop)
- 3. If the Demo Magento homepage doesn't load automatically please visit <http://workshop.dev/>
- 4. Open up the NetBeans IDE (there is a shortcut on the VM Desktop)

Preparations without the Innovate VirtualBox Image:

- 1. Unpack the archive *Magento_Workshop_Full.tgz*. It contains the example modules **Example_MultiOptionCart** and **Example_Brand**.
- 2. Create a database and import the database dump *Magento_Workshop.sql* found in the archive.
- 3. Execute the SQL, replacing "your-domain.dev" with whatever your development domain is:

```
UPDATE core_config_data SET value="http://your-domain.dev/"  
WHERE path LIKE '%base_url';
```
- 4. Edit the settings in the file *app/etc/local.xml* file to point to the imported database.
- 5. Visit the homepage of the Magento Installation to confirm all is working
- 6. Open the Magento Installation in the IDE of your choosing

VirtualBox Image Authentication Credentials:

If you are using the VirtualBox VM you can use the following credentials:

Magento

Magento Frontend URL: `http://workshop.dev/`

Magento Admin URL: `http://workshop.dev/admin`

Magento Admin User: `admin`

Magento Admin Password: `magento123`

MySQL

PhpMyAdmin URL: `http://workshop.dev/phpmyadmin`

MySQL Root User: `root`

MySQL Root Password: `magento123`

Linux

VM User: `magento`

VM Password: `magento123`

Directory Structure

In the VirtualBox Image all files can be found at `/var/www/workshop`. A project has already been created for you in NetBeans.

Inside the workshop directory you have the following sub directories:

/magento

The Magento Application Code as well as the example modules

/onepage-checkout-workshop

The Code for the other Innovate DevCamp Magento Workshop

/performance-workshop/Part One/Code

/performance-workshop/Part Two/Code

The pre-baked Code Snippets for the hands-on Labs

/performance-workshop/Part One/Lab N

/performance-workshop/Part Two/Lab N

The Code as it is before the specified hands-on Lab begins

/performance-workshop/Part One/Lab N finished

/performance-workshop/Part Two/Lab N finished

The Code as it is after the specified hands-on Lab

Workshop Part One:

Example_Brand

Lab 1 - The DB Connection Profiler

Open the brands page

1. In the Browser, click on the **Brands** Category in the top Navigation

Enable the Profiler Display

1. Log in to the Magento admin interface at *http://workshop.dev/admin*
2. Select the page "**System > Configuration**" from the top navigation
3. Select the **Developer** section on the bottom left
4. In the **Debug** panel switch the **Profiler** option to **Yes** and save
5. Reload the front end brands page. Notice the Profiler output at the bottom of the page and that there is no data displayed (yet).

Enable DB Connection Profiling

1. In the IDE, open the file *magento/app/etc/local.xml*
2. Add the a node from code block 01-01 in the node `global/resources/default_setup/connection` after the `<active>` node.

Code Block 01-01:

```
<profiler>true</profiler>
```

```
<model><![CDATA[mysql4]]></model>
<type><![CDATA[pdo_mysql]]></type>
<pdoType><![CDATA[]]></pdoType>
<active>1</active>
<profiler>true</profiler>
</connection>
</default_setup>
</resources>
<session_save><![CDATA[files]]></session_save>
</global>
```

3. Clear the **Configuration Cache** under "**System > Cache Management**"

Test the DB Profiler

Reload the brands page in the front end and notice the Profiler now displays some statistics. The numbers will be different depending on the hardware and software setup you are using.

[\[profiler\]](#)

Memory usage: real: 20185088, emalloc: 19936968

Executed 150 queries in 0.56288003921509 seconds

Average query length: 0.0037525335947673 seconds

Queries per second: 266.48662157068

Longest query length: 0.51635408401489

Longest query:

```
SELECT `main_table`.* FROM `example_brand_entity` AS `main_table`
```

Lab 2 - Collection Caching

Locate the block class.

1. Find the block type by visiting the Admin Panel Page “**CMS > Static Blocks**”.
2. Select the block “**Example Brand List**”
3. Notice the block type *example_brand/list*.

Alternatively you could enable template path hints with block names to find the used block class.

Initialize the Collection Cache

1. Open the file `magento/app/code/local/Example/Brand/Block/List.php`
2. Add the code block 02-01 to the method `_prepareBrandCollection()`

Code Block 02-01:

```
$brands->initCache(Mage::app()->getCache(), 'brandlist', array());
```

```
protected function _prepareBrandCollection(Example_Brand_Model_Resource_Brand_Collection $brands, $limit)
{
    $brands->initCache(Mage::app()->getCache(), 'brandlist', array());
    $brands->addCategories()
        ->addAssociatedWithCategoryFilter()
        ->addAssociateWithSpecialPriceProductFilter()
        ->addCategoryProductCount()
        ->setRandomOrder()
        ->setPageSize($limit);

    return $this;
}
```

Test the Code

Reload the brands page once and notice there is no big difference in speed as the cache entry only was created.

Reload the page again and notice the Profiler no longer lists the collection query as the slowest one.

```
Profiler

Memory usage: real: 20185088, emalloc: 19941504

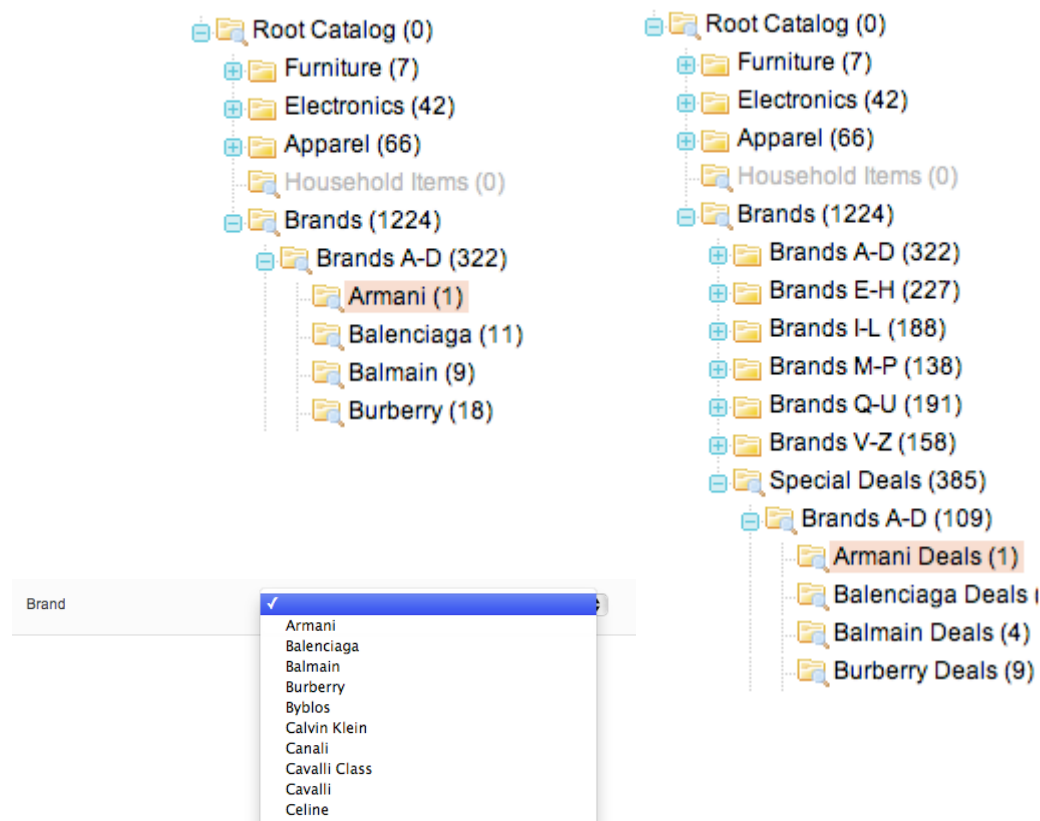
Executed 149 queries in 0.031927824020386 seconds
Average query length: 0.00021428069812339 seconds
Queries per second: 4666.7759100922
Longest query length: 0.00060296058654785
Longest query:
` WHERE (store_id = :store_id) AND (date_from <= :required_date or date_from :
```

Add Cache Invalidation

On the brands category page only brands are listed, which are associated with at least one category and one or more product with a special price.

Assert the current page does not reflect changes once the collection is cached by removing all category associations for a brand

1. Check the brand collection is cached (by checking the profiler output).
2. Choose one of the listed brands (how about the one on the top left)
3. Log in to the Magento admin interface at <http://workshop.dev/admin>
4. Navigate to “**Catalog > Manage Categories**”
5. Select the chosen brand’s category and choose the empty option for the “Brand” Attribute. Save the category change.
6. Do the same for the brand’s category in the “Special Deals” category branch.
7. Reload the brands page and notice that it indeed not change



Add Cache Tags for the Categories

1. Open the file *magento/app/code/local/Example/Brand/Block/List.php*
2. Add the cache tag constants `Mage_Catalog_Model_Category::CACHE_TAG` and `Mage_Catalog_Model_Product::CACHE_TAG` to the collection cache initialization. The whole change is in code block 02-02.

Code Block 02-02:

```
$brands->initCache(Mage::app()->getCache(), 'brandlist', array(  
    Mage_Catalog_Model_Category::CACHE_TAG,  
    Mage_Catalog_Model_Product::CACHE_TAG  
));
```

Test the Code

Since the collection is already cached with no associated code blocks, the cache needs to be cleared once manually for the change to take effect.

1. Go to the **Cache Management** and refresh the “**Collection Data**” cache.
2. Reload the brands page on the front end to regenerate the cache entry.
3. Briefly take note of the current brand order
4. In the admin panel, change any attribute and save the change.
5. Reload the brands page and notice the collection was no longer cached. The collections SELECT is listed in the profiler again and the order of categories changed.
6. Reload the page one more time to assert the collection has been cached again.
7. If you have time go ahead and check it also works for products.

Lab 3 - Entity Cache Tags

Now the collection cache is refreshed every time a category or product is updated, but that is not the case for the brands themselves. If a new brand is added, or an existing one is modified, we still need to update the cache manually.

Add a custom cache tag to the brand entity

1. Open the file *magento/app/code/local/Example/Brand/Model/Brand.php*
2. Add the cache tag constant `CACHE_TAG` with the value 'example_brand' to the head of the class `Example_Brand_Model_Brand`
3. Add the protected var `$_cacheTag` to the class head with the value of the constant (see code block 03-01).

Code Block 03-01:

```
const CACHE_TAG = 'example_brand';

protected $_cacheTag = self::CACHE_TAG;
```

4. Add the brand cache tag to the collection cache initialization in the method `Example_Brand_Block_List::_prepareBrandCollection()` as shown in code block 03-02.

Code Bock 03-02:

```
$brands->initCache(Mage::app()->getCache(), 'brandlist', array(
    Mage_Catalog_Model_Category::CACHE_TAG,
    Mage_Catalog_Model_Product::CACHE_TAG,
    Example_Brand_Model_Brand::CACHE_TAG
));
```

Test the Code

1. Refresh the “**Collection Data**” cache manually.
2. Reload the brands page to regenerate the cache record.
3. Visit the Page “**Catalog > Manage Brands**”.
4. Select a brand, change its name and save it.
5. Reload the brands page and check the result is not from the cache.

Lab 4 - Block Caching

This is an additional possibility to cache data in Magento. Which option to choose depends on the circumstances.

Enable block html caching for the brands page.

1. Open the file *magento/app/code/local/Example/Brand/Block/List.php*
2. Add the protected method `_construct()` as shown in code block 04-01.

Code Block 04-01:

```
protected function _construct()
{
    $this->setCacheLifetime(false);
    $this->setCacheTags(array(
        Mage_Catalog_Model_Category::CACHE_TAG,
        Mage_Catalog_Model_Product::CACHE_TAG,
        Example_Brand_Model_Brand::CACHE_TAG
    ));
    parent::_construct();
}
```

Test the Code

1. Reload the brands page to generate the block cache record.
2. Update a Category, Product or Brand entity through the admin interface.
3. Reload the brands page and check the result is not from the cache.

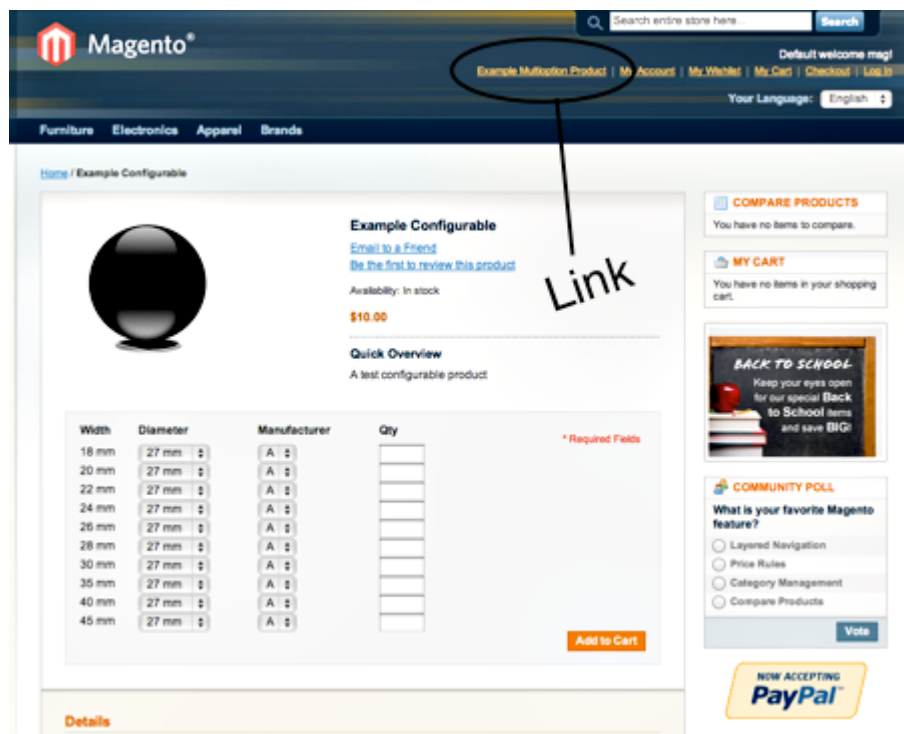
Workshop Part Two:

Example_MultiOptionCart

Lab 05 - The Code Profiler

The issue

1. Visit the Example Multioption Product Page (the link is in the header)
2. Choose some random variants in the selects for three rows.
3. Enter a quantity for those three rows
4. Note the time it takes when you submit the form



Enable the Code Profiler

1. Open the *magento/index.php* file and remove the comment character # on line 64 as shown in the code block 05-01).

Code Block 05-01:

```
require_once $mageFilename;

Varien_Profiler::enable();

if (isset($_SERVER['MAGE_IS_DEVELOPER_MODE'])) {
    Mage::setIsDeveloperMode(true);
}

#ini_set('display_errors', 1);
```

2. Reload the multioption product page on the front end to check on the bottom of the page the code profiler is working.

[Profiler]				
Memory usage: real: 82051072, emalloc: 80337880				
Code Profiler	Time	Cnt	Emalloc	RealMem
mage	5.0599	1	0	0
mage::app::init::system_config	0.0021	1	32,760	0
CORE::create_object_of::Mage_Core_Model_Cache	0.0010	1	358,120	262,144
mage::app::init::config::load_cache	0.0063	1	2,232	0
mage::app::init::stores	0.0247	1	2,018,208	2,097,152
CORE::create_object_of::Mage_Core_Model_Resource_Website	0.0003	1	22,856	0

If the profilerdata is not visible, enable the profiler output as described in the first part of this workshop:

1. Log in to the admin panel at <http://workshop.dev/admin>
2. Visit the page “**System > Configuration**”
3. Select the **Developer** section on the bottom left
4. In the “**Debug**” panel switch the “**Profiler**” option to “Yes” and click Save
5. Reload the multioption product page on the front end to check the profiler is enabled.

Measure performance

1. Add some products to the cart using the form on the multioption page
2. Notice the profiler only displays the data from the cart page, because you are redirected to the *checkout/cart/index* page.
So we need a different way to view the profiling data.
3. Disable the profiler output again in the admin panel at “**System > Configuration > Developer > Debug**”.
4. Keep the call to `Varien_Profiler::enable()` in the `index.php` file enabled.

Log profiling data through an Event Observer

Create the extension **Example_Profiler**

Remember you can use the pre-baked code to copy & paste from if you don't want to type all.

1. Create the module registration file *magento/app/etc/modules/Example_Profiler.xml*
Add the code block 05-02 as the contents of that file.

Code Block 05-02:

```
<?xml version="1.0"?>
<config>
    <modules>
        <Example_Profiler>
            <active>true</active>
            <codePool>local</codePool>
            <depends>
                <Mage_Core/>
            </depends>
        </Example_Profiler>
    </modules>
</config>
```

2. Create the directories *magento/app/code/local/Example/Profiler/etc*
3. Create the file *magento/app/code/local/Example/Profiler/etc/config.xml*
4. Add the code block 05-03 as the contents of that file

Code Block 05-03:

```
<?xml version="1.0"?>
<config>
    <global>
        <models>
            <example_profiler>
                <class>Example_Profiler_Model</class>
            </example_profiler>
        </models>
    </global>
</config>
```

```

        </models>
    </global>
    <frontend>
        <events>
            <controller_front_send_response_after>
                <observers>
                    <example_profiler>
                        <type>model</type>
                        <class>example_profiler/observer</class>
                        <method>controllerFrontSendResponseAfter</method>
                    </example_profiler>
                </observers>
            </controller_front_send_response_after>
        </events>
    </frontend>
</config>

```

5. Create the directory *magento/app/code/local/Example/Profiler/Model* and the file *app/code/local/Example/Profiler/Model/Observer.php* with code block 05-04 as the content.

Code Block 05-04:

```

<?php

class Example_Profiler_Model_Observer
{
    public function
        controllerFrontSendResponseAfter(Varien_Event_Observer $observer)
    {
        // Only write to log if the profiler is enabled
        $timers = $this->_getTimerData();
        if ($timers) {
            $data = $this->_getTableHeadData();
            $data = array_merge($data, $timers);

            $f = fopen('var/log/profiler.log', 'a');
            foreach ($data as $row) {
                fwrite($f, implode(', ', $row) . "\n");
            }
            fclose($f);
        }
    }

    protected function _getTimerData()
    {
        $timers = Varien_Profiler::getTimers();
        foreach ($timers as $name => $timer) {

```

```

    $sum = Varien_Profiler::fetch($name, 'sum');
    $count = Varien_Profiler::fetch($name, 'count');
    $emalloc = Varien_Profiler::fetch($name, 'emalloc');
    $realmem = Varien_Profiler::fetch($name, 'realmem');

    // Filter out entries of little relevance
    if ($sum < .0010 && $count < 10 && $emalloc < 10000) {
        continue;
    }
    $data[] = array($name, number_format($sum, 4), $count,
        number_format($emalloc), number_format($realmem));
}
return $data;
}

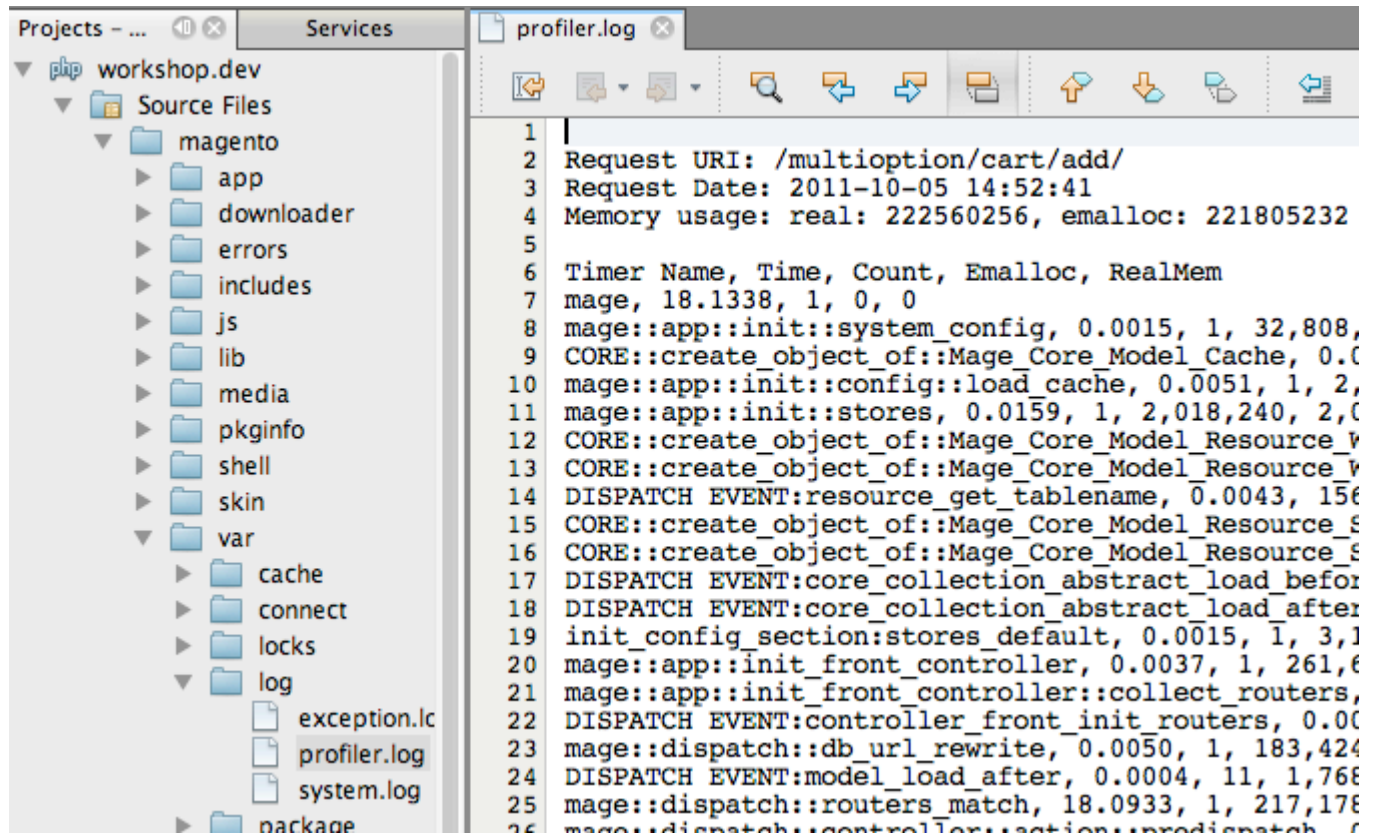
protected function _getTableHeadData()
{
    $data = array();
    $data[] = array();
    $data[] = array('Request URI: ' .
        Mage::app()->getRequest()->getServer('REQUEST_URI'));
    $data[] = array('Request Date: ' . date('Y-m-d H:i:s'));
    $data[] = array('Memory usage: real: ' . memory_get_usage(true),
        'emalloc: ' . memory_get_usage());
    $data[] = array();
    $data[] = array('Timer Name', 'Time', 'Count', 'Emalloc',
        'RealMem');

    return $data;
}
}

```

Test the Code

1. Flush the Magento configuration cache
2. Add some products to the cart using the form on the multioption page
3. Study the profiler data written to the file *magento/var/log/profiler.log*



```
1 |
2 Request URI: /multioption/cart/add/
3 Request Date: 2011-10-05 14:52:41
4 Memory usage: real: 222560256, emalloc: 221805232
5
6 Timer Name, Time, Count, Emalloc, RealMem
7 mage, 18.1338, 1, 0, 0
8 mage::app::init::system_config, 0.0015, 1, 32,808,
9 CORE::create_object_of::Mage_Core_Model_Cache, 0.0
10 mage::app::init::config::load_cache, 0.0051, 1, 2,
11 mage::app::init::stores, 0.0159, 1, 2,018,240, 2,0
12 CORE::create_object_of::Mage_Core_Model_Resource_V
13 CORE::create_object_of::Mage_Core_Model_Resource_V
14 DISPATCH EVENT:resource_get_tablename, 0.0043, 156
15 CORE::create_object_of::Mage_Core_Model_Resource_§
16 CORE::create_object_of::Mage_Core_Model_Resource_§
17 DISPATCH EVENT:core_collection_abstract_load_befor
18 DISPATCH EVENT:core_collection_abstract_load_after
19 init_config_section:stores_default, 0.0015, 1, 3,1
20 mage::app::init_front_controller, 0.0037, 1, 261,6
21 mage::app::init_front_controller::collect_routers,
22 DISPATCH EVENT:controller_front_init_routers, 0.00
23 mage::dispatch::db_url_rewrite, 0.0050, 1, 183,424
24 DISPATCH EVENT:model_load_after, 0.0004, 11, 1,768
25 mage::dispatch::routers_match, 18.0933, 1, 217,178
26 mage::dispatch::controller_section::predispatch /
```


Lab 6 - Formatting the Profiler Data

Currently the profiler output is not really human readable.

Update the observer to log to a text table.

1. Open the file *magento/app/code/local/Example/Profiler/Model/Observer.php*
2. Add the the method `_getDataAsTextTable()` from code block 06-01

Code Block 06-01:

```
protected function _getDataAsTextTable(array $data)
{
    $table = new Zend_Text_Table(array('columnWidths' => array(1)));
    $widths = array();
    foreach ($data as $rowData)
    {
        $row = new Zend_Text_Table_Row();
        foreach ($rowData as $idx => $cell) {
            $width = mb_strlen($cell);
            if (! isset($widths[$idx]) || $widths[$idx] < $width)
            {
                $widths[$idx] = $width;
            }
            $row->appendColumn(new Zend_Text_Table_Column(strval($cell)));
        }
        $table->appendRow($row);
    }
    $table->setColumnWidths($widths);

    return $table->render();
}
```

3. Update the method `controllerFrontSendResponseAfter()` to reflect code block 06-02

Code 06-02:

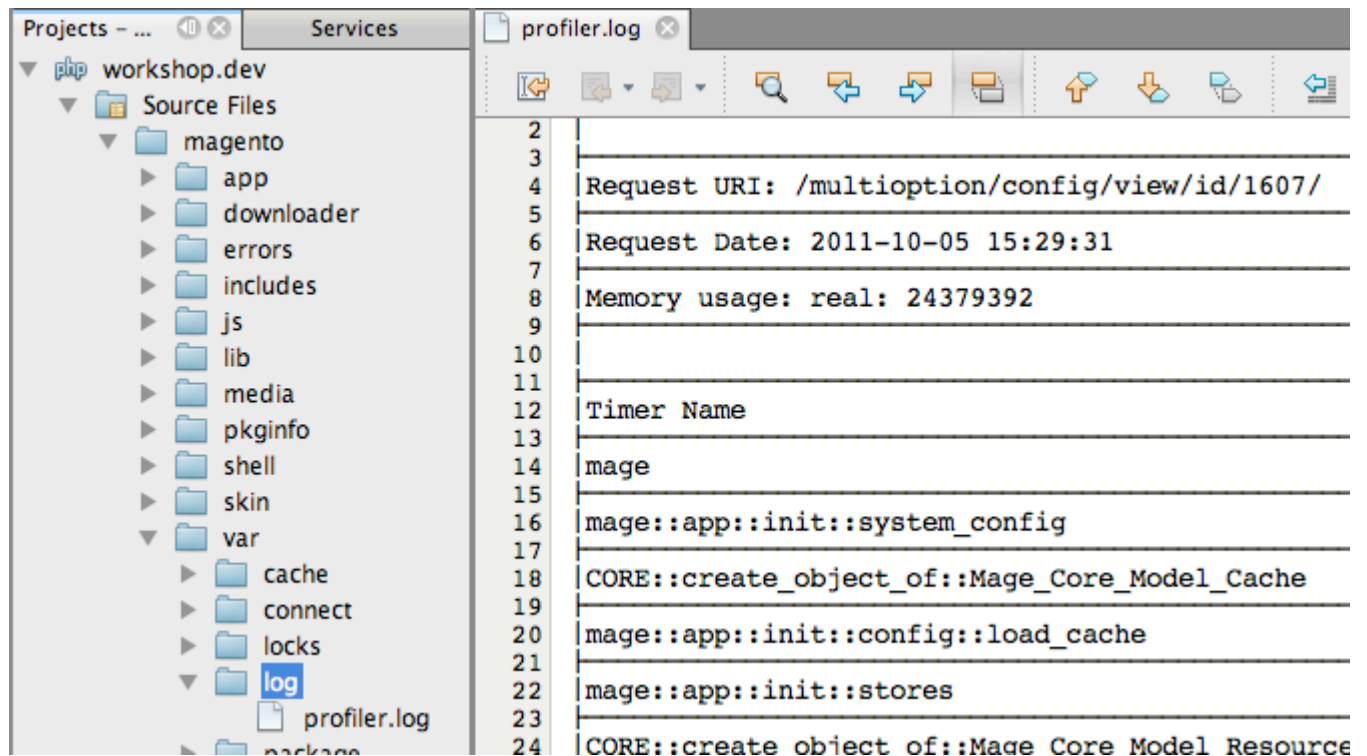
```
public function
    controllerFrontSendResponseAfter(Varien_Event_Observer $observer)
{
    $timers = $this->_getTimerData();
    if ($timers)
    {
        $data = $this->_getTableHeadData();
        $data = array_merge($data, $timers);

        $out = $this->_getDataAsTextTable($data);
        $f = fopen('var/log/profiler.log', 'a');
        fwrite($f, $out);
        fclose($f);
    }
}
```

}

Test the Code

1. Delete the file *magento/var/log/profiler.log*
2. Add some products to the cart using the form on the multioption page
3. Study the changes in the data written to the file *magento/var/log/profiler.log*



The screenshot shows an IDE with a project named 'workshop.dev'. The 'Source Files' tree on the left shows the 'magento' directory structure, including 'var/log' where 'profiler.log' is located. The main editor window displays the contents of 'profiler.log' with line numbers 2 through 24. The log data includes request URI, date, memory usage, and a list of timer names for various Magento processes.

```
2  
3  
4 | Request URI: /multioption/config/view/id/1607/  
5  
6 | Request Date: 2011-10-05 15:29:31  
7  
8 | Memory usage: real: 24379392  
9  
10  
11  
12 | Timer Name  
13  
14 | mage  
15  
16 | mage::app::init::system_config  
17  
18 | CORE::create_object_of::Mage_Core_Model_Cache  
19  
20 | mage::app::init::config::load_cache  
21  
22 | mage::app::init::stores  
23  
24 | CORE::create object of::Mage Core Model Resource
```

Modify Observer to sort Profiling Data by Time

1. Open the file `app/code/local/Example/Profiler/Model/Observer.php`
2. Add the method `_sortTimers()` from the code block 06-03 to the observer class.

Code Block 06-03:

```
protected function _sortTimers(array $a, array $b)
{
    if ($a[1] === $b[1])
    {
        return 0;
    }
    return $a[1] < $b[1] ? 1 : -1;
}
```

3. Add the call `usort(...)` to the method `_getTimerData()` as shown in code block 06-04.

Code Block 06-04:

```
protected function _getTimerData()
{
    $timers = Varien_Profiler::getTimers();
    foreach ($timers as $name => $timer) {
        $sum = Varien_Profiler::fetch($name, 'sum');
        $count = Varien_Profiler::fetch($name, 'count');
        $emalloc = Varien_Profiler::fetch($name, 'emalloc');
        $realmem = Varien_Profiler::fetch($name, 'realmem');

        // Filter out entries of little relevance
        if ($sum < .0010 && $count < 10 && $emalloc < 10000) {
            continue;
        }
        $data[] = array($name, number_format($sum, 4), $count,
            number_format($emalloc), number_format($realmem));
    }
    usort($data, array($this, '_sortTimers'));
    return $data;
}
```

Test the Code

1. Delete the file *magento/var/log/profiler.log*
2. Add some products to the cart using the form on the multioption page
3. Study the changes in the data written to the file *magento/var/log/profiler.log*.
Notice the records are listed in descending order by time.

Timer Name	Time
mage	15.2740
mage::dispatch::routers_match	15.2049
mage::dispatch::controller::action::multioption_cart_add	15.1277
CONFIGURABLE:Mage_Catalog_Model_Product_Type_Configurable::getConfigurableAttributes	14.1614
TTT2:Mage_Catalog_Model_Resource_Product_Type_Configurable_Attribute_Collection::_afterLoad	10.8213
CONFIGURABLE:Mage_Catalog_Model_Product_Type_Configurable::getUsedProducts	10.8183
__EAV_COLLECTION_AFTER_LOAD__	5.9249

Lab 07 - Finding and Fixing the Issue

Now that we have readable profiler data, we can analyze it and implement a solution

Notice the entry in the profiling data from adding the products to the cart with the timer name `CONFIGURABLE:Mage_Catalog_Model_Product_Type_Configurable::getConfigurableAttributes`

Also notice the Count column says the method was called multiple times (the exact number depends on the number of products you added to the cart).

1. Open `magento/app/code/core/Mage/Catalog/Model/Product/Type/Configurable.php`
2. Find the method `getConfigurableAttributes()`
3. Notice the results of the method are cached on the specified product model using the method `setData()`

```
public function getConfigurableAttributes($product = null)
{
    Varien_Profiler::start('CONFIGURABLE:'.__METHOD__);
    if (!$this->getProduct($product)->hasData($this->_configurableAttributes)) {
        $configurableAttributes = $this->getConfigurableAttributeCollection($product)
            ->orderByPosition()
            ->load();
        $this->getProduct($product)->setData($this->_configurableAttributes, $configurableAttributes);
    }
    Varien_Profiler::stop('CONFIGURABLE:'.__METHOD__);
    return $this->getProduct($product)->getData($this->_configurableAttributes);
}
```

4. Open `magento/app/code/local/Example/MultiOptionCart/controllers/CartController.php`
5. Notice a new clone of the product model is used for each call to `$cart->addProduct()`

```
foreach ($options as $params)
{
    $product = clone $protoProduct;
    $cart->addProduct($product, $params);
}
```

So what is happening? The collection of configurable attributes is loaded again for each product.

Implement a Solution

1. Before the `foreach ($options as $params)` loop, add a call to `getConfigurableAttributes()` on the prototype product model as shown in code block 07-01.

Code Block 07-01:

```
try
{
    $protoProduct->getTypeInstance(true)
        ->getConfigurableAttributes($protoProduct);
    foreach ($options as $params)
    {
        $product = clone $protoProduct;
        $cart->addProduct($product, $params);
    }

    $cart->save();
}
```

Test the Code

1. Delete the file *magento/var/log/profiler.log*
2. Add some products to the cart using the form on the multioption page
3. Study the changes in the data written to the file *magento/var/log/profiler.log*.

Timer Name	Time
<u>mage</u>	4.5633
<u>mage::dispatch::routers_match</u>	4.5160
<u>mage::dispatch::controller::action::multioption_cart_add</u>	4.4836
<u>CONFIGURABLE:Mage_Catalog_Model_Product_Type_Configurable::getConfigurableAttributes</u>	3.8622
<u>TTT2:Mage_Catalog_Model_Resource_Product_Type_Configurable_Attribute_Collection::_afterLoad</u>	2.9449
<u>CONFIGURABLE:Mage_Catalog_Model_Product_Type_Configurable::getUsedProducts</u>	2.9436
<u>__EAV_COLLECTION_AFTER_LOAD__</u>	1.5788

4. Disable the profiling by commenting out the call to `Varien_Profiler::enable()` in the *magento/index.php* file when you are done.